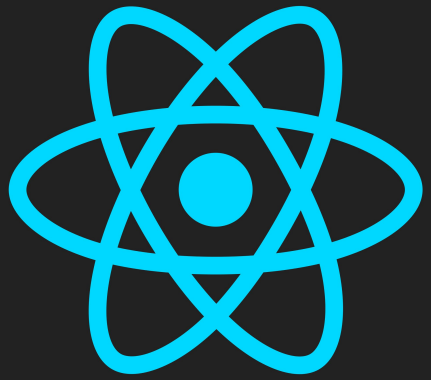
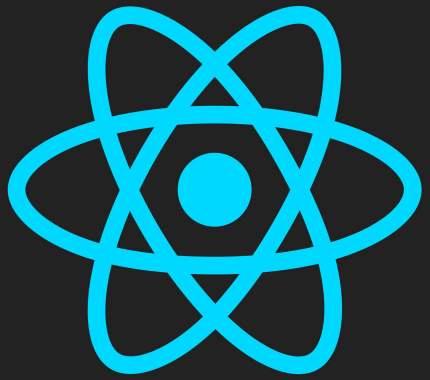


Intro to React



Howdy!

Raul Torres Jr.

Software Developer For Over 8 Years

“Perpetual Junior Developer”

@uz88 | lowbudgetcode.com | www.raul.codes

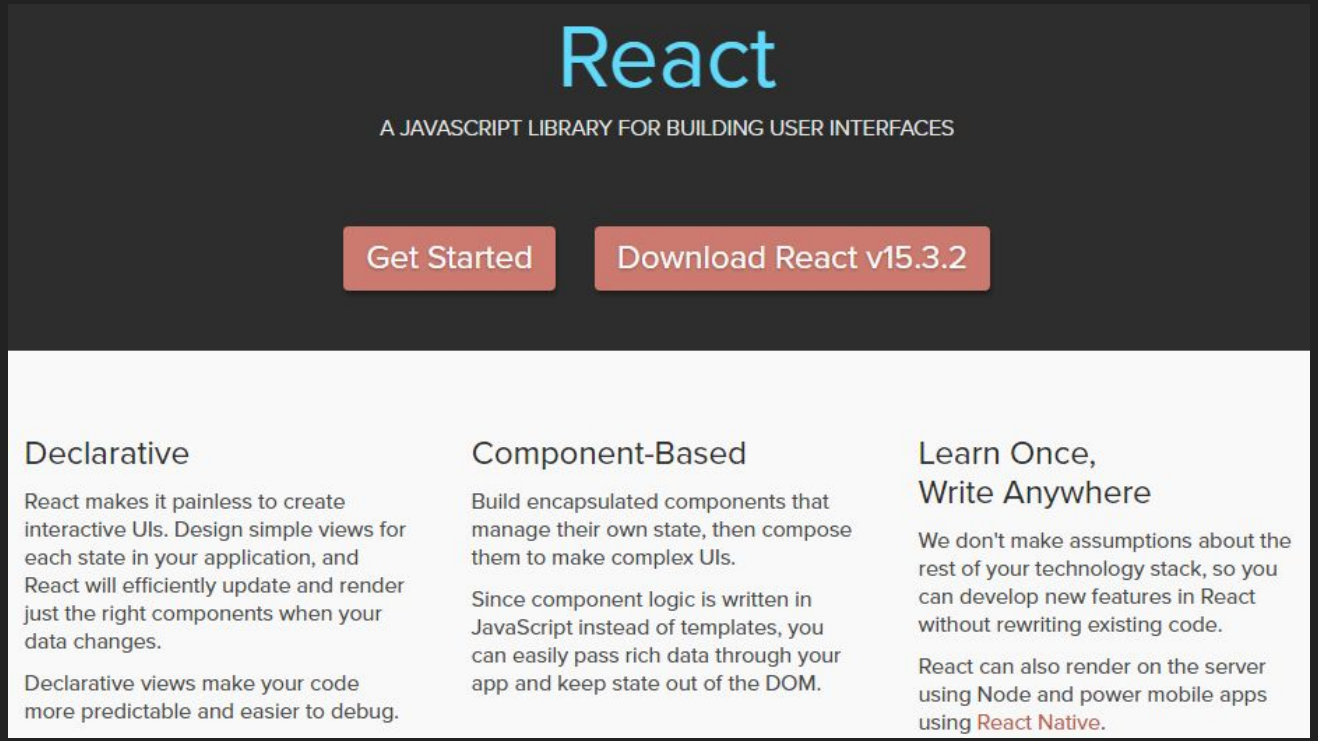
LAUNCH FESTIVAL 2016

call-em-all

Dealertrack

ApexCCTV

What is React?



Who Is Using React?

- Facebook
- Instagram
- Khan Academy
- Netflix
- Visual Studio Team Services / Microsoft Team Foundation Server
- Yahoo!

And many more! https://github.com/facebook/react/wiki/Sites-Using-React

Why Is Netflix Using React?

- “React provides powerful **Component** and Mixin APIs”
- “To reduce the penalties incurred by live DOM manipulation, React applies updates to a **virtual DOM** in pure JavaScript and then determines the minimal set of DOM operations necessary via a diff algorithm.”
- “React enabled us to build JavaScript **UI code that can be executed in both server (e.g. Node.js) and client contexts**.”

Why Is Yahoo Using React

- React implements **one-way reactive data flow**
- **Virtual DOM** allows client and **server side rendering**
- Code in Javascript
- Growing and active community

What Are We Going To Build?

Prerequisites

- Gulp - Task/Build Runner
- Bable - JavaScript Transpiler
- ES6 - Latest JavaScript Version
- Node - JavaScript Runtime Env

Prerequisites

Prerequisites - Just Kidding!!

- ~~Gulp - Task/Build Runner~~
- ~~Bable - JavaScript Transpiler~~
- ~~ES6 - Latest JavaScript Version~~
- ~~Node - JavaScript Runtime Env~~

What Are Components?

Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

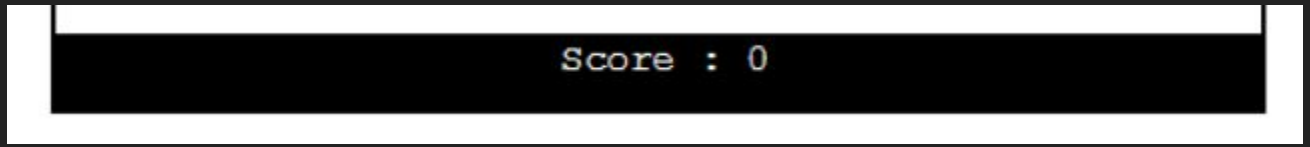
Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

What Components Would You Build?

Components We'll Be Working With

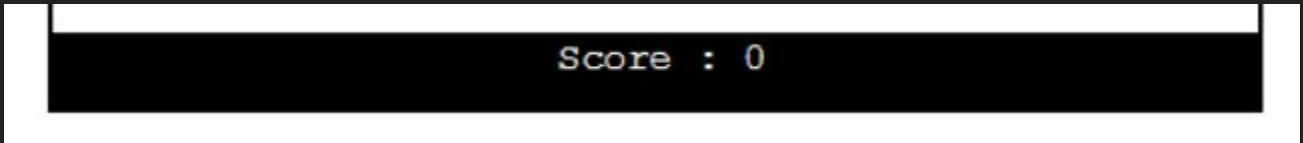
- Main Game
  - Health / Lives
  - Active Game
  - Score

What Is The Score Component's HTML?



Score: 0

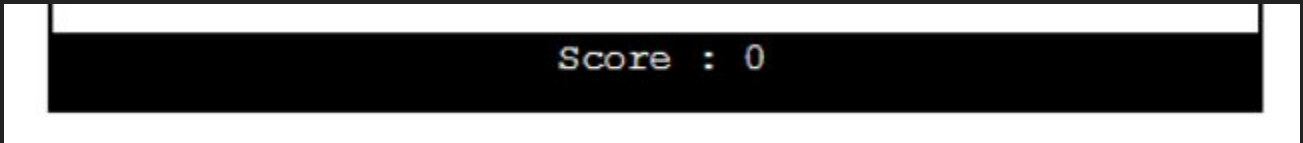
Let’s Make The Score Component With JavaScript




```
var Score = React.createClass({
  render: function() {
    return React.DOM.div( {className : "score"} , "Score: " + this.props.currentScore );
  }
});
```

Score: 0

Let’s Make The Score Component With **JSX**




```
var Score = React.createClass({
  render: function() {
    return ( <div className="score">Score: {this.props.currentScore}</div>);
  }
});
```

Score: 0

JSX > JavaScript > HTML

```jsx
var Score = React.createClass({
  render: function() {
    return ( <div className="score">Score: {this.props.currentScore}</div>);
  }
});
```

```jsx
var Score = React.createClass({
  render: function() {
    return React.DOM.div( {className : "score"} , "Score: " + this.props.currentScore );
  }
});
```

```
<div class="score">Score: 0</div>
```

How Do I Use The Score Component I Just Made?


```
var Score = React.createClass({
  render: function() {
    return ( <div className="score">Score: {this.props.curre
  }
});
```

```
render: function() {
  var activeGameComponent =
    this.componentFactory(this.state.currentComponentName)

  return (
    <div className="MainGame" >
      <Health lives={this.state.lives}/>
      {activeGameComponent}
      <Score currentScore={this.state.score}/>
    </div>
  );
},
```

What Is One-Way Data Flow?

What Is One-Way Data Flow?

- Main Game Sends Data To
  - Health / Lives
  - Active Game
  - Score

How Do We Pass Data To Components?

```
render: function() {
  var activeGameComponent =
    this.componentFactory(this.state.currentComponentName)

  return (
    <div className="MainGame" >
      <Health lives={this.state.lives}/>
      {activeGameComponent}
      <Score currentScore={this.state.score}/>
    </div>
  );
},
```

```
var Score = React.createClass({
  render: function() {
    return ( <div className="score">Score: {this.props.currentScore}</div>);
  }
});
```

Properties A.K.A Props

Using props

Let's create the `Comment` component, which will depend on data passed in from its parent, `CommentList`. Data passed in from a parent component is available as a 'property' on the child component. These 'properties' are accessed through `this.props`. Using props, we will be able to read the data passed to the `Comment` from the `CommentList`, and render some markup:

Code

```
// tutorial4.js
var Comment = React.createClass({
  render: function() {
    return (
      <div className="comment">
        <h2 className="commentAuthor">
          {this.props.author}
        </h2>
        {this.props.children}
      </div>
    );
  }
});
```

Props vs State

What Components Should Have State?

Most of your components should simply take some data from `props` and render it. However, sometimes you need to respond to user input, a server request or the passage of time. For this you use state.

Try to keep as many of your components as possible stateless. By doing this you'll isolate the state to its most logical place and minimize redundancy, making it easier to reason about your application.

A common pattern is to create several stateless components that just render data, and have a stateful component above them in the hierarchy that passes its state to its children via `props`. The stateful component encapsulates all of the interaction logic, while the stateless components take care of rendering data in a declarative way.

Our Props

```
var Score = React.createClass({
  render: function() {
    return ( <div className="score">Score: {this.props.currentScore}</div>);
  }
});
```

The **Score Component does have any states**. We just tell the component how many lives a player has.

The Score Component uses a prop called currentScore.

Keep components Dumb to make component “reusable”

Our States

```
render: function() {
  var activeGameComponent =
    this.componentFactory(this.state.currentComponentName)

  return (
    <div className="MainGame" >
      <Health lives={this.state.lives}/>
      {activeGameComponent}
      <Score currentScore={this.state.score}/>
    </div>
  );
},
```

Our Main Game component has state.

Score increases as a player wins.
Lives decrease as a player loses.
The active game component changes.

The Main Game component sends its state to Score component via the currentScore Prop.

The component **IS NOT** “reusable”

Initial State For Our Main Game

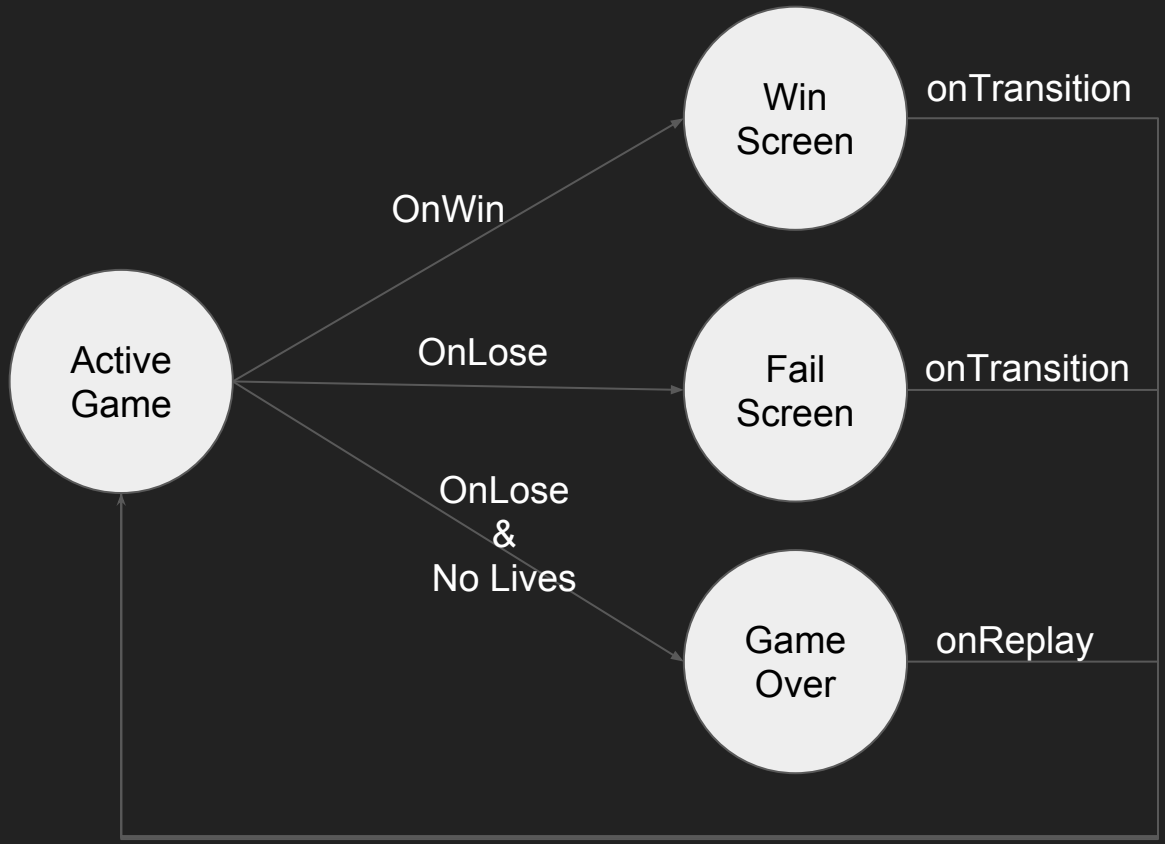
```
getInitialState: function() {
  return {
    lives: 3,
    currentComponentName: 'PressTheButton',
    score: 0
  };
},
```

```
render: function() {
  var activeGameComponent =
    this.componentFactory(this.state.currentComponentName)

  return (
    <div className="MainGame" >
      <Health lives={this.state.lives}/>
      {activeGameComponent}
      <Score currentScore={this.state.score}/>
    </div>
  );
},
```

Warning - Things Are About To Get Real

What Does Our State Machine Look Like?



How Do We Keep One-Way Data Flow?

- Main Game Sends **Functions**
  - Health / Lives
  - Active Game
  - Score

Props Aren’t Just For Data!

```
componentFactory: function(name){
  console.log("The Factory is evaluating", name );

  switch (name) {
    case 'FillTheBarGame':
      return <FillTheBarGame onWinAction={this.onWin} onLoseAction={this.onLose}/>;
    case 'PressTheButton':
      return <PressTheButton onWinAction={this.onWin} onLoseAction={this.onLose}/>;
    case 'EmptyTheBarGame':
      return <EmptyTheBarGame onWinAction={this.onWin} onLoseAction={this.onLose}/>;
    case 'Winner':
      return <Winner onTransitionAction={this.onTransition}/>;
    case 'Loser':
      return <Loser onTransitionAction={this.onTransition}/>;
    case 'GameOver':
      return <GameOver onReplayAction={this.onReplay}/>;
    default:
      return <GameOver onReplayAction={this.onReplay}/>;
  }
},
```

What Happens When We Win?

```
switch (name) {
  case 'FillTheBarGame':
    return <FillTheBarGame onWinAction={this.onWin} onLoseAction={this.onLose}/>;
```

```
onWin: function() {
  console.log('win, setting win screen')
  this.setState({
    lives: this.state.lives,
    currentComponentName: 'Winner',
    previousComponentName: this.state.currentComponentName,
    score: ++this.state.score
  });
```

What About The Virtual Dom?

```
onWin: function() {
  console.log('win, setting win screen')
  this.setState({
    lives: this.state.lives,
    currentComponentName: 'Winner',
    previousComponentName: this.state.currentComponentName,
    score: ++this.state.score
  });
```

What Have We Learned So Far?

- What are React components
- What is JSX
- What makes React components reusable
- What is the difference between props and states
- What is One-Way Data Flow
- How state and prop changes trigger the Virtual Dom
- How to pass functions down to child components to change state from the parent

Dive Into The Code / Questions

https://jsfiddle.net/raultorres88/y16rxx4w/2/ - OR - bit.ly/RT-ReactDemo

- What about Gulp / Babel / ES6 / Node?
- What do the game components look like?
- How do the game components manage their state?
- Why do the game components even have state?
- What is Flux and how does that apply to real world apps?
- You lost me at _____, can you explain it?

Thanks

Raul Torres Jr.

Twitter - @uz88

Blog - lowbudgetcode.com

Me - www.raul.codes

Sources / Footnotes

- A JavaScript library for building user interfaces | React
- Netflix Likes React
- Evolving Yahoo Mail